MySQL2WebDAV

Programmering C
.: HTX 2.2 :.
29-11-2006

af:

Jonas F. _____

Ismail F. _____

Indholdsfortegnelse

Indledning

I dette projekt har vi valgt at arbejde med WebDAV, som bliver grundigt beskrevet i rapporten. Rundt omkring i vores rapport har vi benyttet mange mere eller mindre engelske termer, af frygt at vi en dag skulle deltage i udbredelsen af et dansk term som f.eks. aflusning i stedet for debugging. Vores kildekode og installeret version kan findes med følgende informationer:

URL: ftp://ftp.hotserv.dk

brugernavn: 15272

Password: 3366

phpMyAdmin:https://phpmyadmin.hotserv.dk/

Du kan få adgang til den virtuelle WebDAV ressource gennem din normale fil manager, men en af disse url'er:

- http://webdavtest.hotserv.dk/dav/
- webdav://webdavtest.hotserv.dk/dav/
- dav://webdavtest.hotserv.dk/dav/

Under Windows® kan du gå ind i ”netværkssteder” trykke ”tilføj netværkssted” og skrive http://webdavtest.hotserv.dk/dav/ i guidens url box. Når du har gennemgået resten af guiden kan du åbne den virtuelle WebDAV ressource i Windows Explorer.

Under KDE er webdav tilgændelig som KIO-slave, dvs. at du blot kan åbne Konqueror og skrive webdav://webdavtest.hotserv.dk/dav/ også er du på vej ind på den virtuelle webdav ressource.

Under Gnome er webdav tilgængelig som protokol i Nautilus forkortet som DAV, derfor skriver du blot dav://webdavtest.hotserv.dk/dav/ i Nautilus' adresse linje.

Advarsel, den virtuelle WebDAV ressource er ikke stabil, og implementeringen er kun delvis, så opbevar ikke dokumenter af nogen betydning på WebDAV serveren.

Web-based Distributed Authoring and Versioning

WebDAV er en udvidelse af HTTP-protokollen udarbejdet af en arbejdes gruppe nedsat af IETF (Internet Engineering Task Force). Formålet med WebDAV er at skabe en protokol til redigering af internettet, altså at gøre internettet til at redigerbart medie. Funktionaliteten af WebDAV giver næsten sig selv hvis man kigger nærmere på forkortelsen Web-based Distributed Authoring and Versioning. I første omgang blev V'et Versioning ikke specificeret, specifikationen for versions kontrol i WebDAV protokollen hedder Delta V. På mange måder kan man godt betragte WebDAV som en protokol svarende til FTP (File Transfer Protocol), begge protokoller transportere og redigere data over internettet, altså IP-protokollen (Internet Protocol). Forskellen ligger i at WebDAV er specificeret som nogle udvidelser oven på HTTP-protokollen, hvorimod FTP kører direkte på IP protokollen. Der er selvfølgelig også en masse andre tekniske detaljer i selve implementeringen af protokollerne.

Hvorfor er WebDAV interessant?

Vi har valgt at arbejde med WebDAV fordi protokollen åbner for nogle helt nye muligheder Tidligere har vi måske hørt om GmailFS, et Python script der implementere et virtuelt filsystem oven på FUSE (Filesystem in UserSpacE), med gmail som data lager. Ideen med GmailFS var at du kunne gemme filer i gmail konto, og efterfølgende montere[1] din gmail konto som et filsystem. Mulighederne med virtuelle filsystemer er jo rigtigt mange, vi kommer ind på nogle anvendelses muligheder senere i vores perspektivering.

WebDAV er en rigtig spændende protokol, fordi stort set alle platformer har en WebDAV implementering til stede. På Windows er der en WebDAV implementering gemt i Windows Explorer, på linux, OS X og andre unix varianter har stort set alle respektable fil-manager en WebDAV implementering f.eks. i Gnome VFS eller som kio-slave i KDE. Vi skal altså ikke have brugerne til at installere WebDAV browsere før de kan benytte en WebDAV service, faktisk specificere WebDAV også en "Directory Index" side som giver brugere med en almindelig webbrowser mulighed for gennemse en WebDAV ressource. Det faktum at WebDAV er specificeret som en udvidelse af HTTP-protokollen betyder at stortset alle normale webserver med et serverside sprog, der kan modificere HTTP-headerne kan hoste en WebDAV implementering. Dette inkludere også en almindelig LAMP-server (Linux Apache MySQL Php/Python/Perl).

Vi har altså en protokol der ikke rigtigt bliver udnyttet i dag, hvor både klient og server teknologier er bredt distribueret. Vi har ganske enkelt valgt at beskæftige os med denne protokol, fordi den ikke er særligt ofte benyttet og har et så meget større potentiale. Vi vil kigge på de forskellige udnyttelse muligheder i vores perspektiverings afsnit.

1 Montere eller mount som er det engelsk unix term.

MySQL2WebDAV

Vi har valgt at kalde vores projekt MySQL2WebDAV i mangel på bedre projekt navn. Ideen er at implementere en virtuel WebDAV server, der skaber en abstraktion over en MySQL database. Vores implementering er skrevet i PHP som en klasse nedarvet fra en abstrakt klasse kaldet HTTP_WebDAV_SERVER, der kan installeres fra PEAR (Php Extension and Application Repository), klassen har ikke nogle afhængigheder i PEAR, så man kan bare downloade pakken og benytte den uden om PEAR. HTTP_WebDAV_SERVER implementere det meste af WebDAV protokollen, det vil sige at den håndtere modificering af headere og parser XML. Den klasse vi så har implementeret skal altså nedarve fra HTTP_WebDAV_SERVER og implementere nogle funktioner der videre giver informationer om indholdet af det virtuelle filsystem.

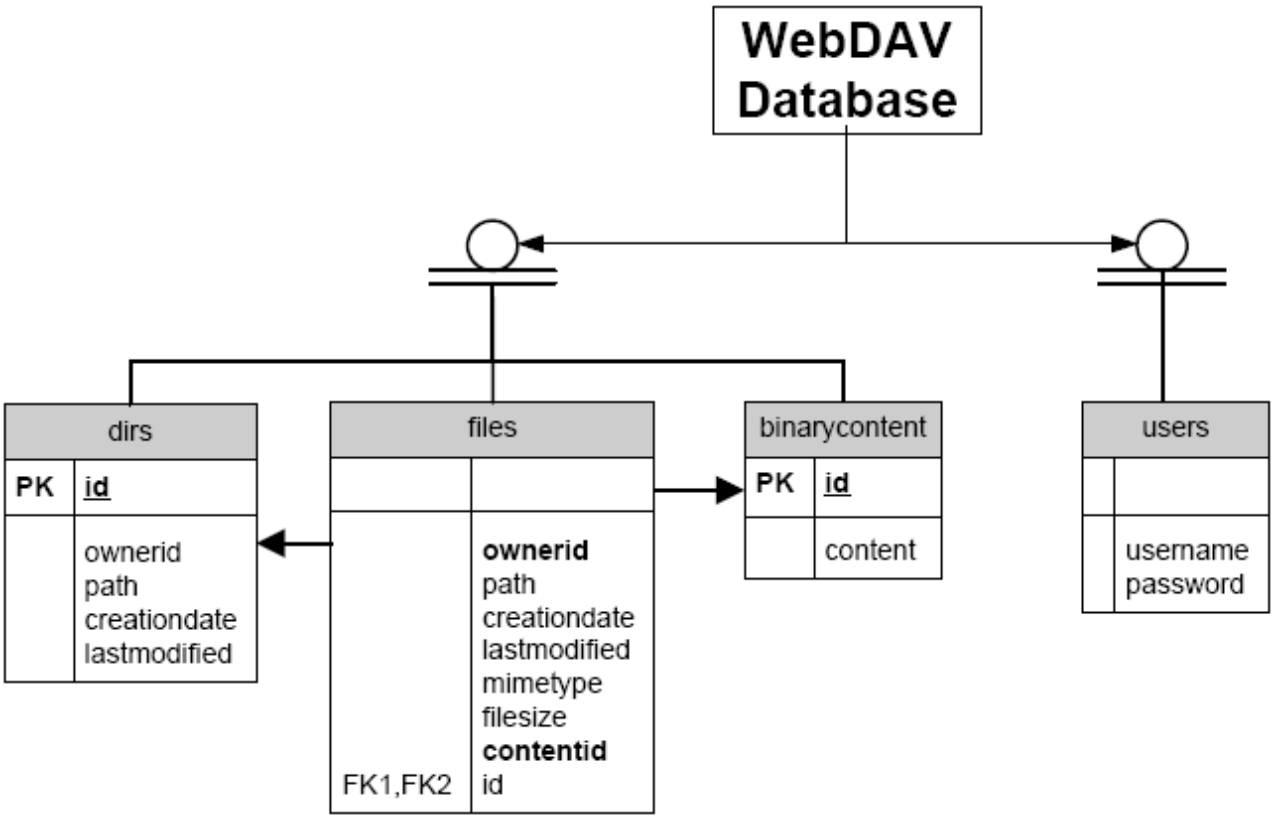
MySQL2WebDAV er mest af alt en proof-of-concept implementering,den kan ikke rigtig benyttes til noget helt konkret. I hvert fald ikke uden grundig omskrivning. Vores implementering er da også alt andet end hundrede procent kompatibel med WebDAV standarterne, som de specificeret af IETF. Det er heller ikke unormalt at en WebDAV implementering ikke lever helt op til standarterne, hverken hvad angår server eller klienter. Hvis vi kort og godt skal opsummere hvad MySQL2WebDAV er, så er den en implementering af en WebDAV server i php, der tager udgangspunkt i HTTP_WebDAV_SERVER ved at nedarve fra denne klasse, som på den måde skaber et virtuelt filsystem som en abstraktion over en MySQL database. Det dog understreges at vi ikke implementere alle funktionerne i protokollen, da dette ligger langt uden for tidsrammerne for dette projekt. Vi regner med at implementere en grundlæggende server, det vil sige at vi regner med delvis understøttelse af PUT, GET og PROPFIND funktionerne som specificeret i RFC2518 kapitel 8 ”HTTP METHODS FOR DISTRIBUTED AUTHORING”.

RFC standarter

RFC eller Request For Comments som de fleste af de standarter vi har refereret til hedder, er standarter der er mere eller mindre under udvikling. Nu kan man godt tænke mange mærkelige ting, når vi bare smider en reference til et RFC standart, en standart der ikke nødvendigvis er helt fastsat endnu. Men disse standarter er faktisk slet ikke så forfærdelige som de lyder, ofte er der blot tale om en 100 siders fil i ren tekst. Jævn kedeligt, men stadigt forståeligt engelsk. Nu er det ikke sådan at vi har siddet og læse alle disse standarter igennem, men de er meget godt som opslagsværker. F.eks. var det meget let af finde ud af hvilken mimetype man skulle tildele en ukendt filtype. IANA (Internet Assigned Numbers Authority) har en liste med RFC standarter for alle registrede mimetypes. På denne liste over RFC standarter kunne man også finde en overordnet standart, der forklarede hvordan mimetype system var opbygget. I den standart stod det også hvad en ukendt mimetype skulle behandles som. Vi han også kigget lidt i det forskellige standarter andre steder, og de kan faktisk være til stor hjælp til tider. Som du sikkert har opdaget referer vi til disse standarter undervejs i rapporten.

WebDAV Database Skema

For at udvikle en databasebaseret Webdav er der selvfølgelig brug for implementering af en database. Vi har valgt at bruge MySQL[2] databasen, da den er gratis, hurtigt, stabil, har god understøttelse i PHP og sidst men ikke menst, vores opgave formuleringen var baseret på det. Vi har implementeret følgende database.

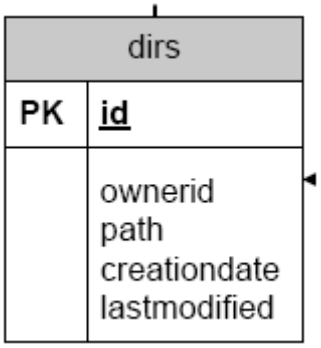


Figur 1: WebDAV Database Diagram

Vi har inddelt databasen i to kategorier. Den første kategori består af et tabel, som bruges til at holde styr på brugerne. Den anden kategori er hovedparten af projektet, hermed den vigtigste del. I denne kategori har vi tre MySQL tabeller, nemlig "dirs", "files" og "binarycontent". Disse tabeller har også relation mellem hinanden. Det er nemlig vist på figur 1. Pilen, som forbinder to tabeller, indikerer, at tabellen, som pilen går fra, er barn (Child) til den tabel, pilen peger på. Relationerne kan findes ved hjælp af nølge felterne, fx "id". Tabellerne i dette kategori har vi beskrivet nøjere i det følgende.

Tabellen "dirs" er til formål at holde styr på mapper på et Webdav-server. Tabellens struktur er vist på figur 2.

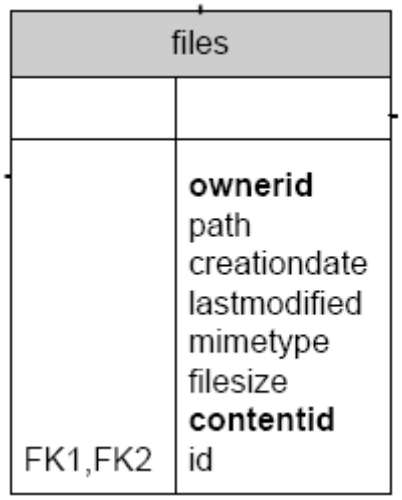
2 http://en.wikipedia.org/wiki/MySQL#Programming_languages



Figur 2: Strukturen for tabel "dirs"

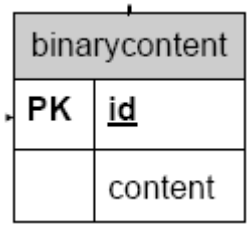
Som det fremgår består tabellen af fem felter, hvor den første felt er tabellens primær nøgle (PK = Primery Key). Desuden er der feltet "ownerid" af typen INT og bruges til at gemme "id" fra forældre-mappen (Parent directory). Der er også feltet "path" af typen text og indeholder den pågældende mappens adresse. Til sidste har vi felterne "creationdate" og "lastmodified" begge to af typen TIMESTAMP og bruges henholdsvis til at holde styr på mappens oprettelsesdato og sidst-redigeret-dato.

Til at holde styre på filerne i en mappe, har vi implenemteret tabellen "files". Tabellens struktur er vist ved hjælp af figur 3.



Figur 3: Strukturen for tabel "files"!

På figuren er der fremhævet to felter i tabellen. Disse to felter -"ownerid" og "contentid" - er til formål at lagre hendholdsvis mappens, som filen befinder sig i, og filens indholds "id" fra henholdsvis "dirs" og "binarycontent" tabeller. Disse to felter er selvfølgelige af typen INT. Felterne "path", "creationdate" og "lasmodified" hardet samme type og funktion, som i tabellen "dirs". I dette tabel er der desuden felterne "mimetype" og "filesize", som henholdsvis er af typerne VARCHAR og INT. Feltet "mimetype" indeholder filens mime-type, som er beskrivet senere. Felten "filesize" indeholder filstørelsen af den pågældende fil. På figur 3 kan man se "FK1, FK2" ved "id". Dette indikerer, at tabellen "files" indeholde to fremmed nøgler (FK = Forgen Key), som er "id".



Figur 4: Strukture for tabel "binarycontent".

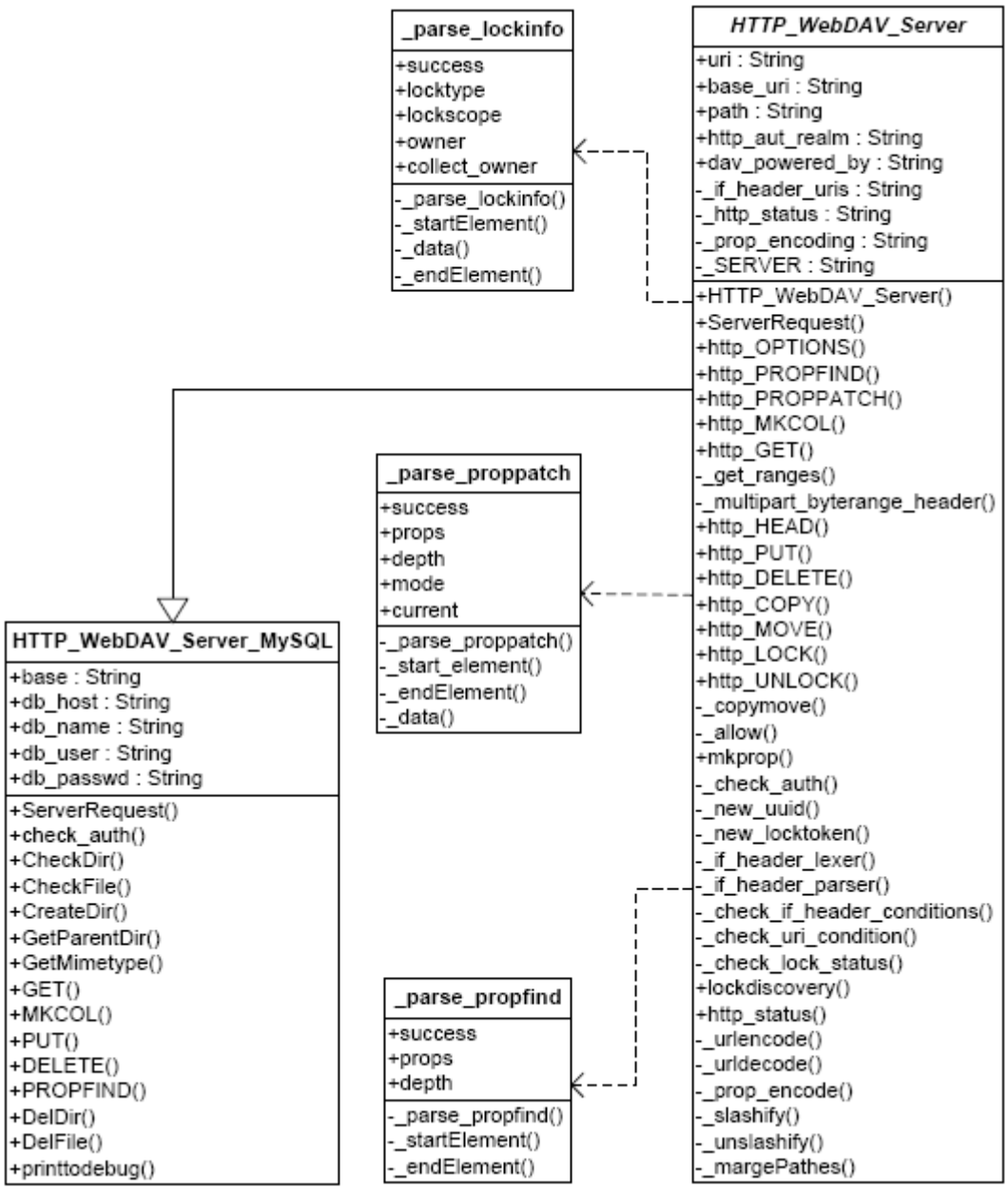
Tabellen "contentbinary" indeholder/lagrer filernes data som binær. Grunden til, at vi har valgt det på denne måde, altså lagre data i et separat tabel i stedet for det samme tabel (files), er optimering. Et MySQL tabel, som indeholder et felt med høj kapacitet, tager langtid at lave forespørgsler på [3]. Derfor er det smart at gøre det på denne måde. Tabellen "binarycontent" har en tilgængelig sktruktur, som er vist på figur 4.

Som det fremgår på figur 4, består tabellen af kun to felter, et felt som primær nøgle "id" af typen INT og et felt til at lagre fil-data i, nemlig "content", som er af typen MEDIUMTEXT[4]. Denne data type har en kapacitet på over 16 milioner tegn.

3 Luke Welling og Luara Thomson: PHP & MySQL Webdevelopment – 3. udgave
4 http://www.resultspk.net/sql/learningsql-chp-2-sect-3.html

Webdav UML Diagram

Webdav er programmeret i PHP, og man har brugt objekt-orientering programmering (OOP). Det vil sige, at Webdav består af nogle Class'er, som bliver arvet af hinanden. Der er ikke forfærdelig mange Class'er, men vi har alligevel valgt, at lave en UML (Unified Modeling Language) diagram over dem for at give overblik og være i stand til at forklare dem. Diagramet er vist på figur 5.



Figur 5: WebDAV UML Diagram

Som det fremgår af figur 5, består Webdav af fem Class'er. Den store Class, HTTP_WebDAV_Server, er en ABSTRACT (abstrakt) Class, som bliver nedarvet af Class'en "HTTP_WebDav_Server_MySQL". "HTTP_WebDav_Server" er hermed forældre-class'en (Parent Class) i dette system, mens "HTTP_WebDav_Server_MySQL" er barn-class'en (Child Class). De resterende Class'er "_parse_lockinfo", "_parse_proppatch" og "_parse_profind" er hjælpere Class'er til "HTTP_WebDav_Server", som er afhængig af dem. Afhængigheden er indikeret med de stiplede pile. Vi har kun modificeret "HTTP_WebDAV_Server_MySQL" for at udvikle et databasebaseret Webdav. De resterende Class'er er i deres grundform og er ikke blevet behandlet på nogen måde I næste afsnit af vi beskrivet "HTTP_WebDAV_Server_MySQL" Class'en nøjer.

Debugging med Litmus

Der er mange komplikationer ved at udvikle en server der implementere en protokol. Normalt kan vi altid få fejl koderne smidt tilfældigt ind i vores HTML output. Når vi normalt udvikler i php kan vi bare åbne applikationen i en webbrowser og se hvad der er galt. Så let er det ikke når man implementere en protokol, for det første kan vi ikke se fejl i vores WebDAV browser, fordi alle fejl bliver behandlet med protokollens egne fejlkoder, uden mulighed for at tilføje ekstra information til fejl koderne. Derfor er vi blevet tvunget til at gå andre vejen for at kunne debugge vores kode. Den ene metode vi har benyttet er at printe variabler til en fil (debug.log), ved hjælp af funktionen printtodebug(). Det hjalp meget og gjorde det betydeligt lettere at fejl finde koden.

Men det var jo ikke helt nok, for vi kunne ikke se hvilke steder vores WebDAV server brød med standarterne. Til dette formål benyttede vi et kommandolinje program kaldet Litmus[5]. Litmus er en WebDAV server protokol kompatibilitets test, som kan teste langt de fleste WebDAV features. Litmus er skrevet i C og skal installeres fra kildekode, der køre på stort set alle moderne Unix varianter Med Litmus kunne vi få et billede over hvilke funktioner der virkede og hvilke der ikke virkede. Desuden fik vi også en debug.log over al kommunikation mellem server (MySQL2WebDAV) og klient (Litmus). Heriblandt de php fejl koder vi havde savnet. Først til allersidst i vores udviklings proces var det muligt at benytte en WebDAV browser til at undersøge hvad der virkede. Derfor har det været et meget spændende projekt, fordi vi først i sidste øjeblik fandt ud af om der overhovedet var noget af det hele der ville virke. Nu skal det igen understreges at vores implementering ikke består hele Litmus testen, men den består faktisk den grundlæggende test med 100% og kun en advarsel. Nedenunder kan man se et opsummeret resultat fra en Litmus test af vores implementering, resultatet af Litmus testen er vedlagt som bilag A:

Test	Fejl	Ikke testet	Advarsler	Korrekt
basic	0	0	3	15
copymove	5	0	2	7
props	3	12	0	11
locks	0	1	1	3
http	0	0	0	4

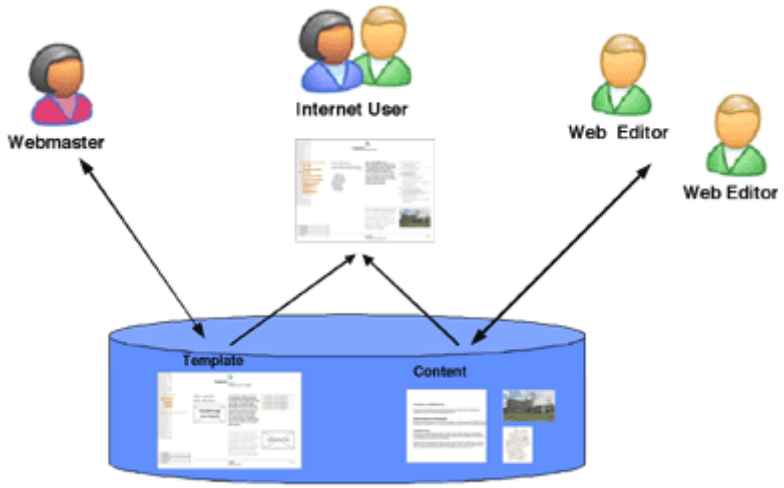
5 http://webdav.org/neon/litmus

Perspektivering

Der findes millioner af store og indholdsrige hjemmeside rundt omkring i verden. Disse websider bliver opdateret dagligt, og der er mange, som arbejder på at holde indholdet på siderne opdateret. Her kan man nævne, BBCworld, CNN, Amazon og millioner af andre lignende sider. Mange af dem, som arbejder på at holde websiderne opdateret, ved ikke alt om web udvikling, design eller noget andet teknisk i forbindelse med websider. Deres opgaver går kun ud på at opdaterer websiderenes indhold, og holder de besøgende på hjemmesiden opdateret med informationer eller noget andet. Disse folk kan være journalister, skolelærere eller have noget helt tradje stilling, som har intet med web udvikling at gøre. Derfor er der brug for et system, som muliggører dette.

Content Management System eller bare CMS er systemet, som bruges siden 1995, til det formål at give brugerne muligheden for at opdatere deres webside uden at vide meget om web udvikling. En diagram over sådan et system kan se således ud:

Det, som vi har opnået med udvikling af en databasbaseret webdav, kan fatisk bruges til noget



Figur 6: Principet i et Content Management System.

vigtig indenfor web udvikling (web development). Det er nemlig et dynamisk, hurtigt og meget brugervenligt Content Management System, som på en måde revolutionere den traditionelle CM Systemer.

Et Content Management System som databasbaseret webdav giver mange muligheder især for de store websider med mange tusinder sider og masse af filer på webserveren. Det er en udfordring for sig selv at holde styr på så mange sider og filer, især når arbejdet udføres af mange bruger, da det er ikke nemt at danner sig et overbliv. Et system som webdav vil give kæmpe overblik over så mange sider og filer, desuden vil tilføjelse, redigering, seltning og vedligeholdelse af filerne blive meget mere hurtigere end CM Systemer, som bruges i øjeblikket.

Da Webdav er udviklet i PHP - i vores tilfælde PHP & MySQL - er den nemt og hurtigt at installere på alle server, som PHP og MySQL kører på. Der skal ikke skrives eller programmeres noget ekstra ved installationen. Det hele er nemt at komme til, da det bygge på den traditionelle filsystem i styresystemet.

Der er vel også mange andre muligheder, hvor man kan integrere teknologien, bort set fra CMS'er. Man kan slevfølgelig også skrive teknologien i andre sprog (webprogrammeringssprog) for at øge dens horisont.

Vurdering

Vi vil vurdere vores projekt som meget godt, det er lykkedes os at implementere en WebDAV server i php, med en MySQL backend. Vi har selvfølgelig ikke understøttelse for hele WebDAV protokollen, men der ligger også langt ud af rammerne for dette projekt. Men vi må også indrømme at dette har været noget af det sværest debugging vi nogensinde har været udsat for. Som tidligere nævt er det utroligt svært at debugge en protokol implementering, nu har vi benyttet os af simple print to debug log og mere avancerede kompatibilitets test som f.eks. Litmus. Men vi kunne også have benyttet os af nogle af de værktøjer der findes til debugging af php. Men alt i alt er vi godt tilfredse med resultatet. Selvom koden stadig har en del bugs, sikkert også nogle af dem der er meget klient specifikke. Vores kode er heller ikke så flot som den kunne være, dette skyldes vores mange ændringer i løbet af vores debugging proces.

Konklusion

På baggrund af vores projekt kan vi konkludere at det er muligt at skabe en WebDAV abstraktion over en lille database med en acceptabel ydelse. Som omtalt i vores perspektivering har dette store muligheder. Vi er godt klar over at vores projekt ikke hjælper verden tættere på udnyttelsen af disse muligheder. Hvis vi skulle have gjort det skulle vi have udviklet et form for layer/lag mellem HTTP_WebDAV_Server og filsystemet eller MySQL databasen. Dette havde ikke været nogen dårlig idé, hvorimod vores nyværende implementering ikke rigtigt kan bruges til noget, fordi den skal omskrives hundrede procent, hvis man vil lave en abstraktion over et andet database skema, f.eks. i forbindelse med integrering i et CMS system.

Bilag

Bilag A – Litmus test:

```
-> running `basic':
 0. init.................. pass
 1. begin................. pass
 2. options............... WARNING: server does not claim Class 2 compliance
    ...................... pass (with 1 warning)
 3. put_get............... WARNING: length mismatch: 0 vs 41
    ...................... pass (with 1 warning)
 4. put_get_utf8_segment.. WARNING: length mismatch: 0 vs 41
    ...................... pass (with 1 warning)
 5. mkcol_over_plain...... pass
 6. delete................ pass
 7. delete_null........... pass
 8. delete_fragment....... pass
 9. mkcol................. pass
10. mkcol_again........... pass
11. delete_coll........... pass
12. mkcol_no_parent....... pass
13. mkcol_with_body....... pass
14. finish................ pass
<- summary for `basic': of 15 tests run: 15 passed, 0 failed. 100.0%
-> 3 warnings were issued.
-> running `copymove':
 0. init.................. pass
 1. begin................. pass
 2. copy_init............. pass
 3. copy_simple........... FAIL (simple resource COPY:
405 Method not allowed)
 4. copy_overwrite........ WARNING: COPY-on-existing fails with 412
    ...................... FAIL (COPY-on-existing with 'Overwrite: T': 405 Metho
d not allowed)
 5. copy_nodestcoll....... WARNING: COPY to non-existant collection '/dav/litmus
/nonesuch' gave '405 Method not allowed' not 409
    ...................... pass (with 1 warning)
 6. copy_cleanup.......... pass
 7. copy_coll............. FAIL (collection COPY `/dav/litmus/ccsrc/' to `/dav/l
```

```
itmus/ccdest/': 405 Method not allowed)
 8. move.................. FAIL (MOVE `/dav/litmus/move' to `/dav/litmus/movedes
t': 405 Method not allowed)
 9. move_coll............. FAIL (collection COPY `/dav/litmus/mvsrc/' to `/dav/l
itmus/mvdest2/', depth infinity: 405 Method not allowed)
10. move_cleanup.......... pass
11. finish................ pass
<- summary for `copymove': of 12 tests run: 7 passed, 5 failed. 58.3%
-> 2 warnings were issued.
-> running `props':
 0. init.................. pass
 1. begin................. pass
 2. propfind_invalid...... pass
 3. propfind_invalid2..... pass
 4. propfind_d0........... pass
 5. propinit.............. pass
 6. propset............... FAIL (PROPPATCH on `/dav/litmus/prop': 405 Method not
 allowed)
 7. propget............... SKIPPED
 8. propextended.......... pass
 9. propmove.............. SKIPPED
10. propget............... SKIPPED
11. propdeletes........... SKIPPED
12. propget............... SKIPPED
13. propreplace........... SKIPPED
14. propget............... SKIPPED
15. propnullns............ SKIPPED
16. propget............... SKIPPED
17. prophighunicode....... SKIPPED
18. propget............... SKIPPED
19. propvalnspace......... SKIPPED
20. propwformed........... pass
21. propinit.............. pass
22. propmanyns............ FAIL (PROPPATCH on `/dav/litmus/prop': 405 Method not
allowed)
23. propget............... FAIL (PROPFIND on `/dav/litmus/prop': 404 Not Found)
24. propcleanup........... pass
25. finish................ pass
```

```
-> 12 tests were skipped.
<- summary for `props': of 14 tests run: 11 passed, 3 failed. 78.6%
-> running `locks':
 0. init.................. pass
 1. begin................. pass
 2. options............... WARNING: server does not claim Class 2 compliance
    ...................... pass (with 1 warning)
 3. precond............... SKIPPED (locking tests skipped,
server does not claim Class 2 compliance)
-> 1 test was skipped.
<- summary for `locks': of 3 tests run: 3 passed, 0 failed. 100.0%
-> 1 warning was issued.
-> running `http':
 0. init.................. pass
 1. begin................. pass
 2. expect100............. pass
 3. finish................ pass
<- summary for `http': of 4 tests run: 4 passed, 0 failed. 100.0%
jopsen@jopsen-laptop:~$
```